

DESCRIPTION

PROGRAM EXECUTION CONTROL DEVICE, PROGRAM EXECUTION CONTROL
METHOD, CONTROL PROGRAM, AND RECORDING MEDIUM

5 Technical Field

The present invention relates to a program execution control device for causing a processor to execute a program composed of sets of bytecodes. More particularly, the present invention relates to a technique for suppressing reduction of execution
10 speed at an initial stage of program execution.

Background Art

A virtual machine is platform-independent software for executing a program. Recent years, virtual machines are
15 implemented on various home appliances equipped with processors such as built-in devices.

One exemplary virtual machine is a Java Virtual Machine (JVM) which interprets and executes Java (Registered Trademark) bytecodes (hereinafter simply "bytecodes"). Detailed
20 specifications of JVM are found in "The Java Virtual Machine Specification, Second Edition" (published by Pearson Education, ISBN 4-89471-356-X).

Generally, bytecodes are generated by a Java compiler, such as Javac, by converting a source program written in the Java
25 language. Bytecodes are stored in so-called class files with extension ".class".

The JVM interprets class files and performs control required to causes a processor to execute methods included in the class

files. A method is a set of bytecodes for execution of a specific procedure and similar to a member function in other programming languages.

Basically, the JVM works as an interpreter sequentially
5 interpreting bytecodes at runtime. As is generally noted, the execution speed of a bytecode program is lower than the execution speed of a native code program directly executable by a processor. Especially, execution of a program in which the same method is repeatedly invoked involves excess overhead because the same
10 method is repeatedly interpreted each time it is invoked.

To reduce the above overhead, there has been suggested a JVM having a Just-In-Time (JIT) compiler. A JIT compiler converts an invoked method into native code if it is judged that the method has not yet been compiled.

15 A JIT improves the execution speed because a processor is allowed to execute previously compiled native code when the same method is invoked for a second time.

Also, there is an extension technology of JIT compiler, which is a technique called Java HotSpot (Registered Trademark)
20 capable of determining a method to be compiled into native code. More specifically, Java Hotspot keeps count of the number of times each method is invoked during program execution and compiles the method into native code when the count exceeds a threshold.

25 JP patent application publication Nos. H4-178734 and S64-62705 disclose program execution control devices having functions of both interpreter and compiler, similarly to the JVM having a JIT compiler.

One problem of JVM having JITs is as follows. The JVM starts compiling a method immediately upon invocation of the method, if the method has not yet been compiled. The resulting native code is executed by a processor when the compilation is done.

5 Thus, the execution speed of an uncompiled method is slower, comparing the interpreter execution of the same method. Reduction in execution speed is especially notable at the initial stage of program execution because more bytecodes need to be compiled.

10 Similarly, the Java HotSpot technology suffers from the slower execution speed when compilation of an uncompiled method is carried out upon invocation. The method can not be executed until the compilation is done.

15 Disclosure of the Invention

In order to address the above problems, the present invention aims to provide a program execution control device for suppressing reduction of execution speed that is conventionally caused when executing an uncompiled method because compilation processing is carried out at the time of
20 executing the method. The present invention also aims to provide techniques related to the program execution control device.

According to one aspect of the present invention, the above aim is achieved by a program execution control device for causing
25 a processor to execute a program composed of one or more sets of bytecodes including a bytecode for invoking a bytecode set. The program execution control device includes: a judging unit operable, for each execution of an invocation bytecode during

execution of the program, to judge whether a bytecode set targeted for invocation is already compiled to native code specific to the processor; a first unit operable, when the target bytecode set is judged to be uncompiled, to instruct the processor so
5 that the target bytecode set is sequentially interpreted and executed, and to issue a request to compile the target bytecode set to native code; a second unit operable, when the target bytecode set is judged to be compiled, to instruct the processor to execute native code resulting from the compilation; and a
10 third unit operable to instruct the processor to compile a bytecode set indicated by a compilation request issued by the first unit, in parallel with the bytecode interpretation and execution by the first unit as well as with the native code execution by the second unit.

15 In another aspect of the present invention, a program execution control method for causing a processor to execute a program composed of one or more sets of bytecodes including a bytecode for invoking a bytecode set. The program execution control method includes: a judging step, for each execution of
20 an invocation bytecode during execution of the program, of judging whether a bytecode set targeted for invocation is already compiled to native code specific to the processor; a first step, when the target bytecode set is judged to be uncompiled, of instructing the processor so that the target bytecode set is
25 sequentially interpreted and executed, and of issuing a request to compile the target bytecode set to native code; a second step, when the target bytecode set is judged to be compiled, of instructing the processor to execute native code resulting from

the compilation; and a third step of instructing the processor to compile a bytecode set indicated by a compilation request issued in the first step, in parallel with the bytecode interpretation and execution in the first step as well as with the native code execution in the second step.

In yet another aspect of the present invention, a control program for causing a processor to execute another program composed of one or more sets of bytecodes including a bytecode for invoking a bytecode set. The control program includes: a judging step, for each execution of an invocation bytecode during execution of the program, of judging whether a bytecode set targeted for invocation is already compiled to native code specific to the processor; a first step, when the target bytecode set is judged to be uncompiled, of instructing the processor so that the target bytecode set is sequentially interpreted and executed, and of issuing a request to compile the target bytecode set to native code; a second step, when the target bytecode set is judged to be compiled, of instructing the processor to execute native code resulting from the compilation; and a third step of instructing the processor to compile a bytecode set indicated by a compilation request issued in the first step, in parallel with the bytecode interpretation and execution in the first step as well as with the native code execution in the second step.

In yet another aspect of the present invention, a recording medium storing a control program for causing a processor to execute another program composed of one or more sets of bytecodes including a bytecode for invoking a bytecode set. The control program includes: a judging step, for each execution of an

invocation bytecode during execution of the program, of judging whether a bytecode set targeted for invocation is already compiled to native code specific to the processor; a first step, when the target bytecode set is judged to be uncompiled, of
5 instructing the processor so that the target bytecode set is sequentially interpreted and executed, and of issuing a request to compile the target bytecode set to native code; a second step, when the target bytecode set is judged to be compiled, of instructing the processor to execute native code resulting from
10 the compilation; and a third step of instructing the processor to compile a bytecode set indicated by a compilation request issued in the first step, in parallel with the bytecode interpretation and execution in the first step as well as with the native code execution in the second step.

15 Here, a bytecode set may be a method or a member function.

With the structure stated above, when a bytecode set (method) invoked has not yet been compiled, the bytecode set is executed by sequential interpretation. Thus, it is ensured to suppress such execution speed reduction that a conventional
20 JIT-enabled JVM suffers from and that is especially notable at an initial stage of the program execution. The execution speed reduction is caused by compiling the method at the time of executing the method. With the above structure, in addition, compilation of uncompiled methods is carried out in parallel
25 with execution of methods. Thus, it is avoided that the execution speed of the overall program becomes slower.

Here, the program execution control device may operate under control of a multitask operating system. The compilation by

the third unit may be executed as a separate task from the bytecode execution by the first unit and the native code execution by the second unit. The tasks of the first and second units may be assigned a higher priority level than a priority level assigned
5 to the task of the third unit.

With the structure stated above, the instruction execution tasks are executed on a priority basis over the compilation task. Thus, it is ensured to suppress execution speed reduction that is conventionally notable at the initial stage of program
10 execution.

Here, the program execution control device may further include a switching unit operable to switch to task execution by the third unit when task execution by the first or second unit is placed in a standby state.

15 With the structure stated above, the compilation task is executed when the instruction execution tasks are put into a standby state. This ensures that the compilation is promptly executed. By promptly executing the compilation, interpreter execution of instructions is required a smaller number of times,
20 which leads to improve execution speed of the overall program.

Here, the program execution control device may further include a request management unit operable to register compilation request information in a storage unit in response to a compilation request issued by the first unit and manage
25 the registered compilation request information. Each piece of compilation request information is used for compiling a bytecode set indicated by a corresponding compilation request. The third unit may instruct the processor to compile each bytecode set

indicated by compilation request information registered in the storage unit, in parallel with the bytecode interpretation and execution by the first unit as well as with the native code execution by the second unit.

5 With the structure stated above, the compilation of bytecode sets are carried out according to compilation request information registered and managed in the storage unit by the request management unit.

10 Here, the request management unit may place pieces of compilation request information in a queue in an order in which corresponding compilation requests are received. The third unit may instruct the processor to compile bytecode sets in order starting from a bytecode set indicated by a first piece of queued compilation request information.

15 With the structure stated above, the compilation of bytecode sets is carried out in the order in which respective compilation requests are received.

20 Here, the request management unit may not register compilation request information in duplicate, if compilation request information for a bytecode set indicated by a compilation request is already registered in the storing unit.

25 With the structure stated above, it is avoided to register a piece of compilation information if the same piece is already registered. Thus, no compilation request information is registered in duplicate.

 Here, the program execution control device may further include: a priority information acquiring unit operable to acquire information showing a priority level of each bytecode

set. The request management unit may include: a specifying subunit operable, in response to a compilation request issued by the first unit, to specify with reference to the acquired priority information a priority level of a bytecode set indicated
5 by the compilation request; a comparing subunit operable to compare the specified priority level with a priority level of each bytecode set indicated by queued compilation request information in the storage unit; and a determining subunit operable to determine a position for placing a new piece of
10 compilation request information for the bytecode set indicated by the compilation request, so that the registered pieces of compilation request information are queued in descending order of priority.

With the structure stated above, bytecode sets having a
15 higher priority level are executed earlier.

Here, the program execution control device may further include: a relational information acquiring unit operable to acquire relational information showing each bytecode set together with all bytecode sets related to the bytecode set;
20 and a detecting unit operable to detect, with reference to the relational information, any bytecode set related to the bytecode set indicated by the compilation request. The request management unit may register compilation request information for the related bytecode set detected by the detecting unit.

25 With the structure stated above, compilation request information is registered additionally for bytecode sets related to a bytecode set that is requested to be compiled.

Here, the program execution control device may further

include a priority information acquiring unit operable to acquire information showing a priority level of each bytecode set. With reference to the acquired priority information, the third unit may instruct the processor to compile bytecode sets indicated
5 by pieces of compilation request information registered in the storage unit, in descending order of priority.

With the structure stated above, bytecode sets having a higher priority level are executed earlier.

Here, the program execution control device may further
10 include: a count recording unit operable to keep a count of compilation requests made to a respective bytecode set when compilation of the bytecode set is repeatedly requested, and records the request count to the storage unit as part of compilation request information for the bytecode set; and an
15 acquiring unit operable to acquire a threshold of request count. The third unit may instruct the processor to compile bytecode sets in order in which respective requests counts exceed the threshold.

With the structure stated above, bytecode sets are compiled
20 in the order in which respective counts of compilation requests exceed the threshold.

Here, the program execution control device may further include: a count recording unit operable to keep a count of compilation requests made to a respective bytecode set when
25 compilation of the bytecode set is repeatedly requested, and records the request count to the storage unit as part of compilation request information for the bytecode set; and an order altering unit operable to compare the respective request

counts and alter positions of pieces of queued compilation request information in descending order of request count.

With the structure stated above, bytecode sets are compiled in descending order of number of compilation requests. Thus,
5 interpreter execution of bytecode sets is required for a less number of times and thus execution speed of the overall program improves.

Here, the request management unit may manage a plurality of queues with different priority levels. The third unit may
10 instruct the processor to compile bytecode sets in order starting from bytecode sets indicated by compilation request information placed in a highest priority queue.

With the structure stated above, depending on queues in which corresponding compilation request information is
15 registered, it is determined which bytecode sets need to be processed on a priority basis and which are not.

Here, the program execution control device may further include: a special request information acquiring unit operable to acquire, prior to execution of the program, information
20 showing a plurality of bytecode sets requested to be compiled. The request management unit may register, in a batch, compilation request information for all bytecode sets shown by the special request information, in a highest priority queue.

With the structure stated above, the special request
25 information is generated to indicate bytecode sets that need to be compiled on a priority basis. This ensures that the bytecode sets are compiled on a priority basis.

Here, the program execution control device may further

include: a second judging unit operable, when the judging unit judges that the target bytecode set is uncompiled, to judge whether the target bytecode set is currently under compilation; and a fourth unit operable, when the target bytecode set is judged
5 to be currently under compilation, to wait until the compilation is done and to subsequently instruct the processor to execute native code resulting from the compilation.

With the structure stated above, when executing an uncompiled bytecode set and the bytecode set is already under
10 compilation, the program execution control device waits until the compilation is done, so that the bytecode set is executed using native code resulting from the compilation.

Here, the program execution control device may further include: a priority information acquiring unit operable to
15 acquire information showing a priority level of each bytecode set; a comparing unit operable to compare priority levels of the bytecode set targeted for the compilation and of the instruction execution task; and a priority altering unit operable to temporarily raises a priority level of the compilation task,
20 when the comparison shows that the priority level of the bytecode set is higher than the priority level of the instruction execution task.

With the structure stated above, when compiling a bytecode set of which priority level is higher than the instruction
25 execution tasks, the priority level of the compilation task is temporarily raised. Consequently, the compilation of that particular bytecode set is carried out promptly.

Brief Description Of The Drawings

FIG. 1 is a view of execution transition between an instruction execution task and a compilation task;

FIG. 2 is a functional diagram of a program execution control device 1 according to an embodiment 1;

FIG. 3 is a view of an exemplary source program written in the Java language;

FIG. 4 is a view of a specific example of compilation request information registered in a queue storage subunit 16 according to the embodiment 1;

FIG. 5 is a view of an example of a compiled method table;

FIG. 6 is a flowchart of an instruction execution task performed;

FIG. 7 is a functional diagram of a program execution control device 1A according to an embodiment 2;

FIG. 8 is a view of a specific example of compilation request information registered in a queue storage subunit 16A according to the embodiment 2;

FIG. 9 is a flowchart of sort processing of compilation request information;

FIG. 10 is a functional diagram of a program execution control device 1B according to an embodiment 3;

FIG. 11 is a view of a specific example of a priority information table 23 acquired from a class file;

FIG. 12 is a view of a specific example of compilation request information registered in a queue storage subunit 16B according to the embodiment 3;

FIG. 13 is a flowchart of compilation request information

placement processing;

FIG. 14 is a functional diagram of a program execution control device 1C according to an embodiment 4;

FIG. 15 is a view of a specific example of a batch registration
5 information table 24;

FIG. 16 is a view of a specific example of compilation request information registered in a queue storage subunit 16C according to the embodiment 4;

FIG. 17 is a functional diagram of a program execution
10 control device 1D according to an embodiment 5;

FIG. 18 is a view of a specific example of a related method table 28;

FIG. 19 is a flowchart of related method detection processing;

FIG. 20 is a functional diagram of a program execution
15 control device 1E according to an embodiment 6;

FIG. 21 is a flowchart of priority inheritance processing;
and

FIG. 22 is a flowchart of a modified instruction execution
20 task.

Best Mode for Carrying Out the Invention

OVERVIEW

A program execution control device according to the present
25 invention may be embodied as a computer device provided with a processor and a storage medium, such as a memory or a hard disk. Specifically, the present invention may be embodied as a mobile phone or a digital broadcast receiver.

The storage medium of the program execution control device stores a multitask operating system (OS), a JVM, class files, and class libraries. A class library is a collection of general-purpose components that are used quite often, such as
5 classes and methods. The program execution control device carries out various functions by a processor operating in accordance with the multitask OS stored in the storage medium and also with the JVM running under control of the multitask OS.

10 When invoking a method during program execution, the program execution control device judges whether the invoked method has been compiled. On judging that the method is uncompiled, the program execution control device executes bytecode associated with the method by interpretation. At the same time, the program
15 execution control device registers a request for compiling the method. On the other hand, on judging that the method has been compiled, native code associated with the method is executed. Compilation of the method relating to the registered request is carried out as a separate task from instruction execution
20 tasks, such as execution of bytecodes by interpretation and execution of native code.

Hereinafter, a task of executing bytecodes or native code is referred to as an instruction execution task, whereas a task of compilation is referred to as a compilation task. A feature
25 of the present invention lies in asynchronous and parallel execution of the instruction execution task and the compilation task.

FIG. 1 is a view of execution transition between the

instruction execution task and the compilation task.

The program execution control device according to the present invention allocates resources of a single processor according to preemptive priority-based task scheduling.

5 The instruction execution task is assigned a higher priority, whereas the compilation task is assigned a lower priority. The processing time is allocated to the higher and lower priority tasks at the ratio of 9:1. It is further controlled that execution is switched to the compilation task immediately upon
10 placement of the instruction execution task into a standby state.

With the above arrangement, the program execution control device of the present invention compiles an uncompiled method not when the method is invoked but when executing the compilation task. Consequently, it is ensured to reduce delay in startup
15 owing to the compilation that is frequently required at the initial stage of program execution, and thus suppressing reduction in execution speed.

Hereinafter, a description is given to embodiments of a program execution control device according to the present
20 invention.

EMBODIMENT 1

Structure 1

FIG. 2 is a functional diagram of a program execution control
25 device 1 according to an embodiment 1.

As shown in the figure, the program execution control device 1 is composed of a class file/class library storage unit 2, an instruction execution unit 3, a compilation request management

unit 4, a native code storage unit 5, a compilation unit 6, a compiled method table 7, and a multitask control unit 19. Note that the figure exclusively shows functional blocks relating to the gist of the present invention. No description is given to non-illustrated functional blocks although those functional blocks are normally provided in a computer implementing a multitask OS and a JVM.

The class file/class library storage unit 2 is a storage medium, and class files and class libraries are stored.

The class files are obtained via an antenna 21 or a network 20.

The instruction execution unit 3 sequentially interprets bytecodes as they are read from the class file/class library storage unit 2, and instructs a processor to execute the interpreted bytecodes. At the time when a previously compiled method is invoked, the instruction execution unit 3 instructs the processor to execute native code associated with the method.

Now, a description is given to functional blocks included in the instruction execution unit 3. The instruction execution unit 3 includes a class load subunit 8, an instruction fetching subunit 9, an instruction interpreting subunit 10, an instruction executing subunit 11, a method-type judging subunit 12, and a native code fetching subunit 13.

The class load subunit 8 loads to a memory a class file or a class library from the class file/class library storage unit 2.

The instruction fetching unit 9 fetches, on an instruction-by-instruction basis, bytecodes associated with

the class file or class library loaded to the memory.

The instruction interpreting subunit 10 interprets fetched bytecode instructions into native code.

5 The instruction executing subunit 11 instructs the processor to execute native code.

If an instruction targeted for execution by the instruction executing subunit 11 is an invocation instruction for invoking a method, the method-type judging subunit 12 judges with reference to the compiled method table 7 whether the method
10 targeted for invocation has previously been compiled. Examples of invocation instructions in Jasmin-syntax assembler code include the following: "invokevirtual", "invokespecial", "invokestatic", and "invokeinterface".

When it is judged that the target method is compiled, the
15 native code fetching subunit 13 fetches native code associated with the method from the native code storage unit 5, and sends the fetched native code to the instruction executing subunit 11. The instruction executing subunit 11 instructs the processor to execute the fetched native code.

20 On the other hand, when it is judged that the target method is uncompiled, the method-type judging subunit 12 issues a compilation request for the method to the compilation request management unit 4. In addition, the instruction fetching subunit 9 fetches a bytecode set associated with the target method
25 on an instruction-by-instruction basis. The instruction interpreting subunit 10 sequentially interprets the fetched bytecode instructions. Finally, the instruction executing subunit 11 instructs the processor to execute the native code

generated by interpreting the bytecode instructions associated with the target method.

5 The compilation management unit 4 receives a compilation request from the method type judging unit 12, and manages the compilation order of methods indicated by received compilation requests. The compilation request management unit 4 includes a registration request receiving subunit 14, a queue control subunit 15, and a queue storage subunit 16.

10 The registration request receiving subunit 14 receives a compilation request from the method type judgment unit 12. By receiving a compilation request, the registration receiving unit 14 acquires information about a method requested to be compiled, such as an identifier identifying the method and a storage address of the bytecode set associated with the method.

15 The queue storage subunit 16 is a storage medium, such as a RAM.

The queue control subunit 15 stores a plurality of pieces of compilation request information into the queue storage unit 16, and manages the stored pieces of compilation request information in a queue. More specifically, upon receipt of a compilation request by the registration request receiving subunit 14, the queue control unit 15 registers, into the queue, compilation request information for the compilation request. Upon receipt, from the compilation unit 6, of an acquisition request for compilation request information, the queue control subunit 15 informs the compilation unit 6 about the method indicated by the first piece of queued compilation request information. When the compilation of the method is done, the

20

25

compilation processing unit 6 informs the queue control subunit 15 about the completion. In response, the queue control subunit 15 deletes the corresponding piece of compilation request information from the queue storage unit 16.

5 At the time of informing the compilation unit 6 about a method indicated by the first piece of queued compilation request information in response to an acquisition request for compilation request information, the queue control unit 15 sets a bit flag included in the compilation request information to show the
10 compilation state of the method. When the flag is set, the method is currently under compilation.

 The queue control subunit 15 performs exclusive control of registration and deletion of compilation request information. That is, the queue control subunit 15 never performs registration
15 and deletion of compilation request information at the same time. In addition, the queue control subunit 15 avoids registering compilation request information if the same piece of compilation request information is already registered in the queue storage subunit 16.

20 The compilation unit 6 is composed of a compilation request acquiring subunit 17 and a method compiling subunit 18. The compilation request acquiring subunit 17 acquires, by issuing an acquisition request, information about a method requested to be compiled from the queue control unit 15. The method
25 compiling subunit 18 compiles a bytecode set associated with the method into native code.

 Native code generated by the method compiling subunit 18 is stored in the native code storage unit 5. Correspondingly,

the compiled method table 7 comes to additionally store a piece of compiled method information that is composed of a method name and a storage address of native code associated with the method. Upon completion of the compilation, the compilation request
5 acquiring subunit 17 informs the queue control subunit 15 that the method has now been compiled.

The multitask control unit 19 is a function of the multitask OS, and allocates, by priority-based task scheduling, the processor resources to the instruction execution task assigned
10 to the instruction execution unit 3 and the compilation task assigned to the compilation unit 6, respectively.

Data 1

Now, a description is given to various data.

15 First, a description is given to a program executed by the program execution control device 1.

FIG. 3 is a view of an exemplary source program written in the Java language. Note that no description is given to the methods "m1" and "m2" because these methods are not particularly
20 relevant to the present invention. Bytecodes generated by converting the source program shown in the figure is contained in a class file A. The class file A is stored in the class file/class library storage unit 2 of the program execution control device 1.

25 During execution of a class file by the program execution control device 1, a method is invoked with an invocation instruction contained in the class file. In the case of the program shown in FIG. 3, first, the method "init" is invoked

once. Next, the method "m1" is invoked ten times, and then the "m2" method is invoked five times.

Now, a description is given to compilation request information.

5 FIG. 4 is a view of a specific example of compilation request information registered in the queue storage subunit 16.

Each piece of compilation request information is composed of a pointer to the storage location of a subsequent piece of compilation request information, method information, and a bit
10 flag showing the compilation state. That is, each piece of compilation request information is linked to another with the pointer, thereby forming a queue.

The method information is composed generally of an identifier identifying the method requested to be compiled, and
15 the storage address of a bytecode set associated with the method.

The bit flag indicates the compilation state of the method. When the bit flag is set, the method is now in the process of compilation. When the bit flag is not set, compilation of the method has not started yet. In the figure, the method is
20 indicated either as "Under Compilation" or "Uncompiled" for the simplicity sake.

In the figure, the compilation request information for the method "init" is placed at the top of the queue, followed by the compilation request information for the method "m1" and then
25 by the compilation request information for the method "m2". In this example, when receiving from the compilation unit 6 an acquisition request for information about a method to be compiled, the queue control subunit 15 provides information about the

method "init" to the compilation unit 6. In response, the compilation unit 6 compiles the method "init". Upon completion of the compilation, the compilation unit 6 informs the queue control subunit 15 that the method "init" has now been compiled.

5 On receiving the notification that the method "init" is compiled, the queue control subunit 15 deletes the piece of compilation request information for the method "init" from the queue storage subunit 16. In addition, the queue control subunit 15 updates the pointer to the first piece of queued compilation
10 request information, so as to indicate the piece of compilation request information for the method "m1".

Now, a description is given to the compiled method information.

FIG. 5 is a view of an example of the compiled method table
15 7. Each piece of compiled method information is composed of an identifier identifying a compiled method (such as a method name) and the storage address of associated native code.

Operation 1

20 Now, a description is given to operation of the program execution control device 1.

FIG. 6 is a flowchart of the instruction execution task performed by the instruction execution unit 3.

First, the instruction executing subunit 11 causes the
25 processor to execute a fetched native code instruction (step S1).

If the instruction targeted for execution is a program end instruction (step S2: YES), processing is terminated. If the

target instruction is not a program end instruction (step S2: NO), a step S3 is performed.

In the step S3, if the target instruction is an invocation instruction (step S3: YES), a step S4 is performed. In the step
5 S3, on the other hand, if the target instruction is not an invocation instruction (step S3: NO), processing returns to the step S1.

In the step S4, with reference to the compiled method table 7, the method-type judging subunit 12 judges whether a method
10 targeted for invocation has previously been compiled. If the target method is compiled (step S4: YES), a step S5 is performed. If the target method is uncompiled (step S4: NO), a step S6 is performed.

In the step S5, the native code fetching subunit 13 fetches
15 native code associated with the target method from the native code storage unit 5; and sends the fetched native code to the instruction executing subunit 11. Then, processing returns to the step S1.

In the step S6, the method-type judging subunit 12 issues
20 a compilation request to the compilation request management unit 4. In a step S7 that follows, the instruction fetching subunit 9 fetches bytecodes associated with the target method, and the instruction interpreting subunit 10 interprets the fetched bytecodes. Then, processing returns to the step S1.

25 Separately, the compilation unit 6 executes the compilation task. On allocation of the processor resources by the multitask control unit 19, the compilation unit 6 compiles a bytecode set associated with a method, acquired from the queue control subunit

15 in response to an acquisition request that the compilation unit 6 has issued.

Next, a specific example of operation is described with reference to FIGs. 2, 4, and 5. First, consideration is given to the case where the instruction executing subunit 11 shown in FIG. 2 instructs the processor to execute an invocation instruction for the method "m2". In this case, the method-type judging subunit 12 judges whether the method "m2" has previously been compiled, with reference to compiled method table 7 shown in FIG. 5.

The compiled method table 7 stores information identifying the method "m2" as a compiled method, together with the storage address of associated native code. Accordingly, the method-type judging subunit 12 judges that the method "m2" is compiled, and thus instructs the native code fetching subunit 13 to fetch the associated native code. The fetched native code is then sent to the instruction executing subunit 11 and executed.

Considering now the case where the instruction executing subunit 11 instructs the processor to execute an invocation instruction for a non-illustrated method "m3". In this case, since the method "m3" is not listed in the compiled method table 7 shown in FIG. 5, the method-type judging subunit 12 judges that the method "m3" is uncompiled and thus issues a compilation request for the method "m3" to the compilation request management unit 4. Separately, the method "m3" is executed by sequentially interpreting associated bytecodes on an instruction-by-instruction basis, i.e. by so-called interpreter execution.

The compilation request management unit 4 registers the compilation request information for the method "m3" at the end of the queue shown in FIG. 4. That is, the compilation request information is placed subsequent to the compilation request information for the method "m2".

EMBODIMENT 2

The following describes a program execution control device of an embodiment 2 of the present invention. There is a case where a compilation request is made for the same method of which compilation request information is already registered in the queue. In such a case, the program execution control device of the embodiment 2 keeps count of compilation requests made to respective methods and sorts, based on the request counts, pieces of compilation request information that are initially queued in the registration order. As a result, the pieces of compilation request information are queued in the descending order of request count.

With this arrangement, it is ensured that a method that is more frequently invoked is compiled earlier than a less frequently invoked method. Therefore, the program execution speed improves.

The following describes the program execution control device of the embodiment 2, with respect to its structure, data, and operation. Note that description overlapping that of the program execution control device 1 of the embodiment 1 is not repeated here.

Structure 2

FIG. 7 is a functional diagram of a program execution control device 1A according to the embodiment 2.

The program execution control device 1A differs from the
5 program execution control device 1 in that a queue control subunit 15A is additionally provided with an adder 22.

The adder 22 holds a request count of each method as part of a corresponding piece of compilation request information. In response to a compilation request for the method of which
10 compilation request information has already been registered, the adder 22 increments the request count of that method by "1". Note that the increment is performed provided that compilation of the requested method is not yet started at the time of receiving the compilation request. The queue control subunit 15A then
15 updates the corresponding piece of compilation request information with the incremented request count.

Data 2

Next, a description is given to a specific example of
20 compilation request information registered in a queue storage subunit 16A of the embodiment 2.

FIG. 8 is a view of a specific example of compilation request information registered in the queue storage subunit 16A. The compilation request information shown in the figure differs from
25 that of the embodiment 1 in the following points. According to the embodiment 2, each piece of compilation request information includes two pointers, one to the immediately preceding piece of compilation request information and the other

to the immediately subsequent piece. In addition, each piece of compilation request information is provided with an area for storing the request count.

FIG. 8 shows pieces of compilation request information that
5 are sorted in descending order of request count.

Operation 2

Next, a description is given to the sort processing of the queued compilation request information.

10 FIG. 9 is a flowchart of the sort processing of compilation request information performed by the queue control subunit 15A.

First, upon receipt of a compilation request by the registration request receiving subunit 14, the queue control subunit 15A refers to the first piece of queued compilation
15 request information (step S11).

Next, the method indicated by the received compilation request is compared with the method indicated by the method currently referred to. If the two methods match (step S12: YES), a step S13 is performed. If the two methods do not match (step
20 S12: NO), a step S17 is performed.

In the step S17, it is judged whether all the pieces of compilation request information have already been referred to. If all the pieces are referred to (step S17: YES), a step S19 is performed. If not (step S17: NO), a step S18 is performed.

25 In the step S18, in accordance with the pointer of the compilation request information currently referred to, the queue control subunit 15 now refers to an immediately subsequent piece of queued compilation request information, and moves onto a step

S12.

In the step S19, the queue control subunit 15A registers, at the end of the queue, compilation request information for the method indicated by the received compilation request.

5 In the step S13, the queue control subunit 15A judges, with the bit flag of the method indicated by the compilation request information currently referred to, whether compilation of the method has not yet been initiated or is already underway. If the compilation is underway (step S13: NO), the sort processing
10 is terminated. If the compilation is not yet initiated (step S13: YES), a step S14 is performed.

In the step S14, the queue control subunit 15A updates the request count of the compilation request information currently referred to, by incrementing by "1". Next, in the step S15,
15 the queue control subunit 15A compares the request count of an immediately preceding piece of queued compilation request information with the updated request count to see which holds a larger value.

If the comparison shows that the updated request count holds
20 a larger value than the request count of the immediately preceding piece of queued compilation request information (step S15: YES), a step S16 is performed. If not (step S15: NO), the sort processing is terminated.

In the step S16, the queue control subunit 15A alters the
25 positions of the immediately preceding piece of compilation request information with the compilation request information of which request count has been updated. That is, the queue control subunit 15A modifies the pointer information of the two

pieces of compilation request information. In addition, the queue control subunit 15A modifies the pointer information of pieces of compilation request information preceding and subsequent to the two pieces. This completes the sort
5 processing.

EMBODIMENT 3

A program execution control device according to an embodiment 3 acquires a priority information table from a class
10 file. The priority information table stores the priority level of each method. With reference to the acquired priority information table, the program execution control device identifies a priority level of each method to be compiled and determines the order of compilation request information in a
15 queue according to the priority levels.

The above arrangement ensures that a higher priority method is compiled earlier than a lower priority method, so that the program execution speed improves.

The following describes the program execution control
20 device according to the embodiment 3, with respect to its structure, data, and operation. Note that the description overlapping the description of the program execution control device 1 is not repeated here.

25 Structure 3

FIG. 10 is a functional diagram of a program execution control device 1B according to the embodiment 3.

The program execution control device 1B differs from the

program execution control device 1 in that a priority information table 23 is acquired, and that the queue control subunit 15B determines, based on the priority level of associated method, the position in the queue for placing compilation request
5 information.

Data 3

Next, a description is given to various data.

FIG. 11 is a view of a specific example of the priority
10 information table 23 acquired from a class file. The priority information table 23 shows each method with a priority level assigned thereto. The priority level is an index of priority given to the method in the order to be compiled, and a larger value indicates a higher priority. In descending order of
15 priority, the methods "m3", "init", "m2", and "m1" are placed in the stated order. Note in the present embodiment, the methods have different priority levels.

FIG. 12 is a view of a specific example of compilation request information registered in a queue storage subunit 16B. As shown
20 in the figure, compilation request information having a higher priority is placed at the front of the queue.

Operation 3

Next, a description is given to compilation request
25 information placement processing by the queue control subunit 15B.

FIG. 13 is a flowchart of the compilation request information placement processing by the queue control subunit

15B.

First, upon receipt of a compilation request by the registration request receiving subunit 14B, the queue control subunit 15B refers to the first piece of queued compilation request information (step S21).

Next, the queue control subunit 15B compares the method indicated by the received compilation request, with the method indicated by the compilation request information currently referred to. If the two methods match (step S22: YES), the compilation request information placement processing is terminated. If the two methods do not match (step S22: NO), a step S23 is performed.

In the step S23, the queue control subunit 15B compares the priority level of the method for which compilation request information is to be registered, with the priority level shown by the compilation request information currently referred to. As a result of the comparison, if the method currently under registration has a higher priority level (step S23: YES), a step S24 is performed. On the other hand, if the compilation request information currently referred to shows a higher priority level (step S23: NO), a step S25 is performed.

In the step S25, it is judged whether all the pieces of compilation request information have already been referred to. If all the pieces are referred to (step S25: YES), a step S26 is performed. If not (step S25: NO), a step S27 is performed.

In the step S27, in accordance with the pointer of the compilation request information currently referred to, the queue control subunit 15B now refers to an immediately subsequent piece

of queued compilation request information, and goes back to the step S22.

In the step S26, the queue control subunit 15B registers, at the end of the queue, a new piece of compilation request information for the method indicated by the received compilation request. Then, the compilation request information placement processing is terminated.

In the step S24, the queue control subunit 15B registers a new piece of compilation request information at a position immediately preceding the compilation request information currently referred to. Then, the compilation request information placement processing is terminated.

Consider, for example, the case of registering compilation request information for the method "m3", into the queue of compilation request information shown in FIG. 12. The priority level of the method "m3" is "2". In this case, as a result of the compilation request information placement processing, compilation request information for the method "m3" is determined to be placed between pieces of compilation request information for the methods "m2" and "m1". More specifically, the queue control subunit 15B updates the pointers of the compilation request information for the methods "m2" and "m1", so that both the pointers show the storage location of the compilation request information for the method "m3". As a result, the compilation request information for the method "m3" is placed at the determined position in the queue.

EMBODIMENT 4

A program execution control device according to an embodiment 4 registers compilation request information in one of three queues having a high priority, a medium priority, and a low priority, and compiles methods in order starting from methods indicated by compilation request information in the high priority queue. More specifically, compiled first is the method indicated by the first piece of compilation request information registered in the high-priority queue. After compiling all methods indicated by pieces of compilation request information in the high-priority queue, the method indicated by the first piece of compilation request information registered in the medium-priority queue is compiled. After compiling all methods indicated by pieces of compilation request information in the medium-priority queue, the method indicated by the first piece of compilation request information in the low-priority queue is compiled.

In the high-priority queue, specific pieces of compilation request information are registered in a batch at the time of OS startup. In the middle-priority queue, pieces of compilation request information for prioritized methods are registered according to the priority information table. The low-priority queue is for the remaining pieces of compilation request information registered neither in the high-priority queue nor in the middle-priority queue.

The following describes the program execution control device of the embodiment 4, with respect to its structure, data, and operation. Note that description overlapping that of the program execution control devices of the embodiments 1-3 are

not repeated here.

Structure 4

FIG. 14 is a functional diagram of a program execution control device 1C according to the embodiment 4.

The program execution control device 1C differs from the program execution control device 1, with respect to the following three points. First, a queue control subunit 15C manages three queues having different priority levels. Second, a batch registration unit 25 and a batch registration information table 24 are additionally provided. The batch registration information table 24 stores method information showing the names and storage locations of methods to be compiled with priority. Upon OS startup, the batch registration unit 25 registers, in a queue storage subunit 16C, compilation request information of the methods listed in the batch registration information table 24. As mentioned above, each piece of compilation request information for those methods is placed in the high-priority queue. Finally, the sort processing and the compilation request information placement processing described in the embodiments 2 and 3 are employed.

Data 4

Next, a description is given to various data.

FIG. 15 is a view of a specific example of the batch registration information table 24. Methods listed in the batch registration information table 24 include methods specified by a user to be compiled, and methods expected to be frequently

invoked based on a result of program execution simulation.

FIG. 16 is a view of a specific example of compilation request information registered in the queue storage subunit 16C. As shown in the figure, the queue storage subunit 16C stores the compilation request information in three queues: a first queue having a high priority, a second queue having a medium priority, and a third queue having a low priority. The first queue stores compilation request information for the methods "m4", "m5", and "m6", which are listed in the batch registration information table 24 shown in FIG. 15.

The second queue stores compilation request information for methods to which priority is assigned, and the compilation request information is placed in order of priority. The third queue stores compilation request information of methods other than the ones placed in the first and second queues, and the order of queued compilation request information is dynamically changed through the sort processing.

EMBODIMENT 5

A program execution control device according to an embodiment 5 acquires a related method table from a class file. With reference to the related method table, the compilation unit 6 detects any method related to a method targeted for compilation. If any related method is detected, a registration request for the detected method is issued.

The following describes the program execution control device of the embodiment 5, with respect to its structure, data, and operation. Note that description overlapping that of the

program execution control device 1 of the embodiment 1 is not repeated here.

Structure 5

5 FIG. 17 is a functional diagram of a program execution control device 1D according to the embodiment 5.

The program execution control device 1D differs from the program execution control device 1 in that a related method detection unit 26 is additionally provided.

10 The related method detection unit 26 includes a detecting subunit 27. The detecting subunit 27 acquires a related method table 28, and detects from the acquired related method table 28 any method related to the method informed by the compilation request acquiring subunit 17. If any related method is detected,
15 the related method detecting unit 26 assumes that a compilation request for the method is issued and thus issues a compilation request for each detected method to the registration request receiving subunit 14D.

20 Data 5

FIG. 18 is a view showing a specific example of the related method table 28.

As shown in the figure, the method "init" is related to the methods "m1" and "m2", whereas the method "m3" is related
25 to the methods "m4" and "m5".

Operation 5

Next, a description is given to the operation for related

method detection processing. The related method detection processing starts with detection of a related method and ends with registration of a compilation request for the related method detected.

5 FIG. 19 is a flowchart of the related method detection processing performed by the related method detecting unit 26.

First, in response to a notification from the compilation request acquiring subunit 17 about a method requested to be compiled, the detecting subunit 27 detects any related method
10 with reference to the related method table 28 (step S31).

If any related method is detected (step S32: YES), a step S33 is performed. If no related method is detected (step S32: NO), the related method detection processing is terminated.

In the step S33, the related method detecting unit 26 assumes
15 that a compilation request for the detected related method is made, and issues to the registration request receiving subunit 14D, a compilation request for the detected related method. The processing is then terminated.

Here, the processing is more specifically described with
20 reference to FIG. 18. On receiving a notification about the method "init" from the compilation request acquiring subunit 17, the related method detecting unit 26 detects a method related to the method "init", with reference to the related method table 28. As shown in FIG. 18, the method "init" is related to the
25 methods "m1" and "m2", so that the related method detecting unit 26 issues compilation requests for the methods "m1" and "m2" to the registration request receiving subunit 14D.

EMBODIMENT 6

A program execution control device according to an embodiment 6 performs compilation request information placement processing, based on the priority level of a respective method, as described in the embodiment 3. In addition, the program execution control device compares the priority level of a method to be compiled with the priority level of the instruction execution task. If the comparison shows that the priority level of the method is higher, priority inheritance processing is performed so as to make the priority level of the compilation task equal to that of the method.

The following describes the program execution control device of the embodiment 6, with respect to its structure, data, and operation. Note that description overlapping that of the program execution control device 1B of the embodiment 3 is not repeated here.

Structure 6

FIG. 20 is a functional diagram of a program execution control device 1E according to the embodiment 6.

The program execution control device 1E differs from the program execution control device 1B of the embodiment 3 in that a priority inheritance unit 29 is additionally provided.

The priority inheritance unit 29 includes functional blocks of a priority comparing subunit 30 and a task priority altering subunit 31.

Upon receipt of a notification about the priority level of the method of which compilation is to be initiated, the priority

comparing subunit 30 compares the priority levels of the method and of the instruction execution task.

If the comparison result shows that the priority level of the method targeted for compilation is higher, the task priority
5 altering subunit 31 instructs the multitask control unit 19E to raise the priority level of the compilation task to make it equal to the priority level of the method.

The multitask control unit 19E alters the priority level of the compilation task according to the instruction from the
10 task priority altering subunit 31.

Operation 6

Next, a description is given to the priority inheritance processing.

15 FIG. 21 is a flowchart of the priority inheritance processing.

First, on receiving a notification from the compilation request acquiring subunit 17 about the priority level of method of which compilation is to be initiated, the priority inheritance
20 unit 29 compares the priority levels of the method and of the instruction execution processing (step S41).

If the comparison shows that the priority level of the method is higher (step S42: YES), the task priority altering subunit 31 instructs the multitask control unit 19E to alter the priority
25 level of the compilation task to make it equal to the priority level of the method (step S43). On the other hand, if the priority level of the execution task is higher (step S42: NO), the processing returns to the step S41.

Suppose, for example, the priority levels of the instruction execution task and of the compilation task default to "6" and "2", respectively. The priority level of method targeted for compilation is "7". In this case, the comparison by the priority
5 comparing subunit 30 shows that the priority levels of the method and of the instruction execution task are $7 > 6$. That is, the priority level of the method is higher, so that the task priority altering subunit 31 instructs the multitask control unit 19E to alter the priority level of the compilation task to make it
10 equal to the priority level of the method. In accordance with the instruction, the multitask control unit 19E raises the priority level of the compilation task to "7", which is higher than the default priority level "6" of the instruction execution task. Consequently, the processor resources are allocated to
15 the compilation task on a priority basis.

SUPPLEMENTAL REMARKS

It is naturally appreciated that the present invention is not limited to the specific embodiments described above.
20 Various modifications including the following may be made.

(1) According to the embodiment 1, when the method-type judging subunit 12 judges that compilation of the method to be invoked is not done, the method is immediately interpreted and executed. Yet, in the case where the invoked method is under
25 compilation, it is applicable to wait for the method to be compiled and execute the resulting native code. Operation for such processing is described below with reference to FIG. 22.

FIG. 22 is a flowchart of an instruction execution task

in which execution of a method under compilation is postponed until the compilation is done.

First, the instruction executing subunit 11 instructs the processor to execute a fetched instruction (step S51). If the
5 instruction targeted for execution is a program end instruction (step S52: YES), the processing is terminated. If not (step S52: NO), a step S53 is performed.

In the step S53, if the target instruction is an invocation instruction (step S53: YES), a step S54 is performed. If not
10 (step S53: NO), the processing returns to the step S51.

In the step S54, the method-type judging subunit 12 judges, with reference to the compiled method table 7, whether the method targeted for invocation has previously been compiled. If the target method has been compiled (step S54: YES), a step S55 is
15 performed. If not (step S54: NO), a step S56 is performed.

In the step S55, the native code fetching subunit 13 fetches native code associated with the target method from the native code storage unit 5, and sends the fetched native code to the instruction executing subunit 11. The processing then returns
20 to the step S51.

In the step S56, the method-type judging subunit 12 instructs the queue control subunit 15 to check the queue to see if the target method is currently under compilation. If the target method is currently under compilation (step S56: YES),
25 the method-type judging subunit 12 waits until the compilation is done (step S57). If not (step S56: NO), the method-type judging subunit 12 issues a compilation request to the compilation request management unit 4 (step S58). In a step

S59 that follows, the instruction fetching subunit 9 fetches bytecodes associated with the method and the instruction interpreting subunit 10 interprets the fetched bytecodes. The processing then returns to the step S51.

5 (2) FIG. 1 shows the execution transition between one execution task and one compilation task. Yet, there may be a plurality of instruction execution tasks and compilation tasks. In such a case, different priority levels may be assigned to the tasks.

10 In the above embodiments, the instruction execution task is assigned a higher priority than the compilation task. When there is a plurality of compilation tasks, it is applicable to assign, as an exception, a higher priority to a specific compilation task than the instruction execution tasks.

15 In the embodiment 4, for example, there are three queues with high, medium, and low priority levels. Here, it is applicable to assign a higher priority level to the compilation task of compiling methods indicated by compilation request information registered in the first queue, than the priority
20 level of the instruction execution task.

 (3) The program execution control devices according to the above embodiments are described as executing the tasks by pseudo-parallel processing. Yet, the program execution control device according to the present invention may concurrently
25 execute the tasks with a multiprocessor or a plurality of separate processors.

 (4) Application of the present invention is not limited to a program execution control device implementing a JVM with

a JIT compiler. The present invention is also applicable to any program execution control device implementing a virtual machine having a compiling function for intermediate code.

(5) In the above embodiments, the order of methods for compilation is managed with queues. Yet, it is not essential part of the program execution control device of the present invention to manage compilation order in queues. For example, the order of compilation may be determined by obtaining a threshold specific to each program or class file, and methods are compiled in the order in which the respective request counts exceed the threshold. Alternatively, it is applicable to compile, at predetermined time intervals during program execution, a highest priority method at the time among methods for which compilation request information is registered. Alternatively, it is applicable to compile, each time the processor resources are allocated to the compilation task, a highest priority method at the time among methods for which compilation request information is registered.

(6) In the embodiment 4, three queues having different priority levels are employed. Yet, the number of queues is not limited to three, and two or four queues may be employed, for example.

(7) The program execution control device according to the present invention has a garbage collection function that a JVM generally provides. The garbage collection function is carried out as a garbage collection task, in parallel with the instruction execution task and the compilation task. It is applicable to perform garbage collection to discard generated native code that

will never or rarely be used.

(8) It is applicable to provide a control program for causing a device with a program execution function to perform the processing steps (shown in FIGs. 6, 9, 13, 19, 21, and 22) performed by the program execution control devices according to the embodiments described above. Such a control program may be distributed in form of a recording medium or via various communications lines. Examples of such a recording medium include an IC card, an optical disc, a flexible disk, and a ROM. Distributed control programs may be put to use by being installed into a device having a ROM. By executing the control program, the device carries out the function equivalent to that performed by the program execution control devices of the above embodiments.

Industrial Applicability

A program execution control device according to the present invention is applicable as a virtual machine implemented on various electronic appliances having processors.